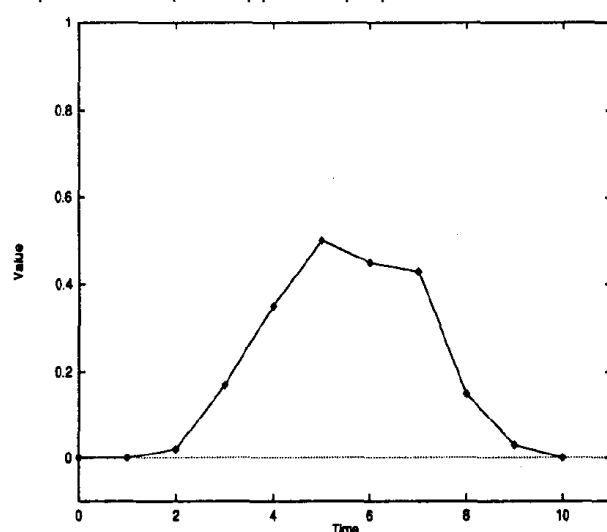


1、时间序列描述问题

SDL (Shape Definition Language) 是对时间序列中形状进行描述的语言，通过 SDL 用户可以进行模糊化的查询。也就是说可以不用知道原始数据的细节，而只需要输入整体的形状特征就可以查询结果。SDL 定义了八类基本转变符号，如下表所示。

Symbol	Description	<i>lb</i>	<i>ub</i>	<i>iv</i>	<i>fv</i>
up	slightly increasing transition	.05	.19	anyvalue	anyvalue
Up	highly increasing transition	.20	1.0	anyvalue	anyvalue
down	slightly decreasing transition	-.19	-.05	anyvalue	anyvalue
Down	highly decreasing transition	-1.0	-.19	anyvalue	anyvalue
appears	transition from a zero value to a non-zero value	0	1.0	zero	nonzero
disappears	transition from a non-zero value to a zero value	-1.0	0	nonzero	zero
stable	the final value nearly equal to the initial value	-.04	.04	anyvalue	anyvalue
zero	both the initial and final values are zero	0	0	zero	zero

假设时间序列为 $H=(0\ 0\ .02\ .17\ .35\ .50\ .45\ .43\ .15\ .03\ 0)$ ，如下图，则相应的转变序列 (Transition Sequence) 为 (zero appears up up down stable Down down disappears)。



基于上述符号，可以定义基本的形状，语法描述为：

(shape name(parameters) descriptor)

例如尖峰 (spike) 可以定义为

(shape spike() (concat Up up down Down))

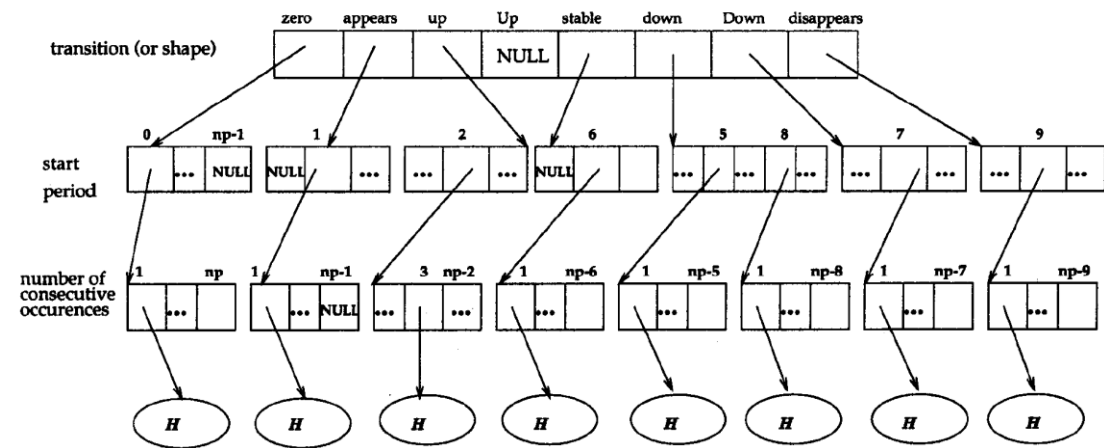
这样所有的符号定义都可以表示为基本的形状。

对于复杂的形状，还可以对基本形状进行组合定义派生形状。有种派生运算：

- 1) any: 形状有多种选择，只要一种匹配即可，语法为 (any P1 P2 ... Pn)。
- 2) concat: 对形状进行连接，语法为 (concat P1 P2 ... Pn)，当时间序列包含所有的基本形状且顺序相同，则认为相匹配。例如，(concat up up up (any stable down) (any stable down) (any down Down))，这样的序列与 $\{H[2, 8]\} = \{.02\ .17\ .35\ .50\ .45\ .43\ .15\}$ 是一致的。
- 3) 多次出现运算 (exact, atleast, atmost)
- 4) 包含运算 in: 指定长度的时间序列中是否包含某些形状，语法为 (in length shape-occurrences)，如 (in 5 (and (noless 2 (any up Up) (nomore 1 (any down Down)))))) 表示 5 个间隔内至少有两个 up (或 Up)，最多一个 down (Down)，这正好对应 $\{H[2, 7]\}$ 。

关于 SDL 作者提到如何建立索引加速形状的查询。索引结构分为四层，最顶层是符号名为索引的数组，大小对应符号的数目。第二层则是一组由第一个符号出现位置构成的数组，每个数组的大小由时间序列长度决定。第三层则由相关符号出现的最大次数为索引，最后一层则

是具体的时间序列，整个索引结构如下图。



最后论文给出了如何通过索引实现形状匹配。

这篇论文很大程度上可以作为我们时间序列形状搜索的基本实现方法，但是有部分内容还需要扩展。对于两个时间序列来说，通常要进行比较，直接的方法是将两个时间序列做差值转换为一个时间序列。如果用 SDL 建立模型只能描述形状信息，但是丢失了差异信息，比如说 $A > B$ 还是 $A < B$ 无法刻画。而且直接做差值会丢失数据原有的形状特征，不利于检索。因此 SDL 只适合于描述单个时间序列的检索，但是对于两个时间序列之间关系的检索无法描述，需要扩展。

2、特征抽取问题

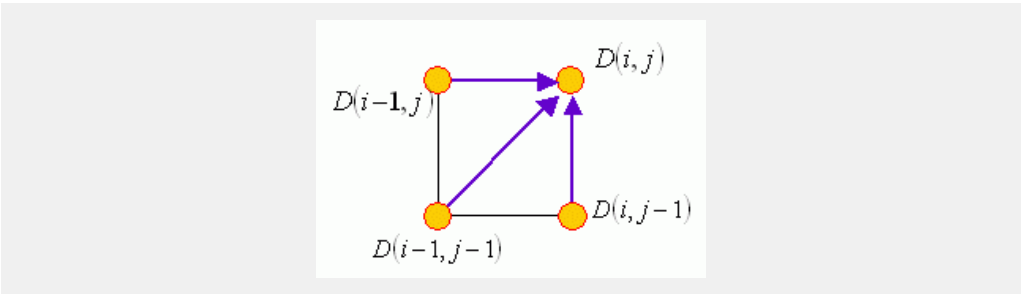
时间序列通常非常长，直接对原始数据进行检索比较费时，如果能够根据需要事先对数据进行特征抽取，去除不必要的信息则能减少数据量。DTW (Dynamic Time Warping) 是目前运用最多的时间序列特征抽取算法。

对于时间序列，欧氏距离存在着其很明显的缺陷，比如说，序列 A: 1,1,1,10,2,3，序列 B: 1,1,1,2,10,3，如果用欧氏距离 $\text{distance}[i][j] = (b[j] - a[i]) * (b[j] - a[i])$ 来计算的话，总的距离和应该是 128，应该说这个距离是非常大的，而实际上这个序列的图形是十分相似的。DTW 基于动态规划的思想，最早用于解决语音序列中发音长短不一的模板匹配问题，简单地说就是通过构建一个邻接矩阵寻找最短路径和。

数学上可以定义为一类优化问题：

假设有两个向量 t 和 r ，长度分别是 m 和 n ，那么 DTW 的目标，就是找到一组路径 $\{(p_1, q_1), (p_2, q_2), \dots, (p_k, q_k)\}$ ，使得经由上述路径的“点对点”对应距离和 $S_i = \sum_{k=1}^i |t(p_i) - r(q_i)|$ 为最小。同时此路径必须满足下列条件：

- 1) 端点关系: $(p_1, q_1) = (1, 1)$, $(p_k, q_k) = (m, n)$ ，即“头对头，尾对尾”。
- 2) 局部关系: 假设最经路径上任一点可以表示为 (i, j) ，那么其前一点路径只有三种可能: $(i-1, j)$, $(i, j-1)$, $(i-1, j-1)$ ，局部关系定义了路径的连续性，而且也规定了 t 的任何一个元素至少对应一个 r 的元素，反之亦然。

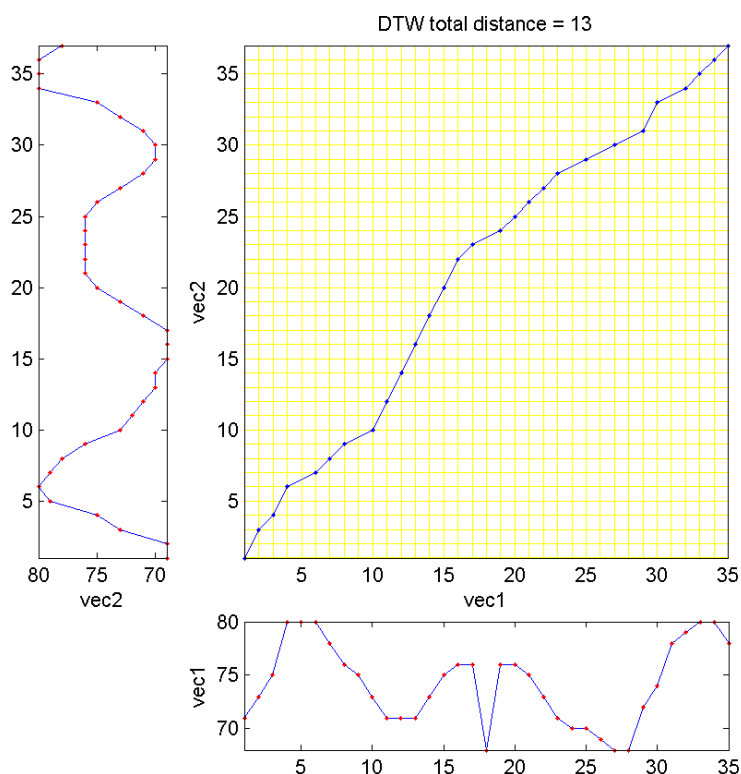


根据动态规划的原理，可以将 DTW 描述成下列三个步骤：

- 1) 目标函数定义：定义 $D(i, j)$ 是 $t(1:i)$ 和 $r(1:j)$ 之间的 DTW 距离，对应的最佳路径是由 $(1, 1)$ 走到 (i, j) 。
- 2) 目标函数的迭代： $D(i, j) = |t(i) - r(j)| + \min\{D(i-1, j), D(i-1, j-1), D(i, j-1)\}$ ，其端点条件为 $D(1, 1) = |t(1) - r(1)|$
- 3) 最终结果： $D(m, n)$ 。

实例：

```
vec1=[71 73 75 80 80 80 78 76 75 73 71 71 71 73 75 76 76 68 76 76 75 73 71 70 70 69 68 68  
72 74 78 79 80 80 78];  
vec2=[69 69 73 75 79 80 79 78 76 73 72 71 70 70 69 69 69 71 73 75 76 76 76 76 76 75 73 71  
70 70 71 73 75 80 80 80 78];
```



对于 DTW 来说，能够很好地解决时间序列特征匹配问题，但是对于大规模的时间序列来说计算复杂度仍然是一个问题。近年来一直有论文开展这方面的研究，如 **DTW-D: Time Series Semi-Supervised Learning from a Single Example**(SIGKDD 2013), **A Novel Approximation to Dynamic Time Warping allows Anytime Clustering of Massive Time Series Datasets** (SDM 2012)。

本周还对 NBA2013 年总决赛的数据做了一点尝试，通过条件匹配在 7 场比赛中寻找在比分落后时能够反超比分的球员，结果如下图，从中可以看出所有队员中 James 效率值最高，而对于马刺来说 Duncan 和 Leonard 对球队贡献基本相同，Parker 决赛中并没有其他比赛中发挥那么好。

